SONDe, Self-Organizing Replica Placement in Large-Scale Dynamic Systems

Vincent Gramoli, Anne-Marie Kermarrec, Erwan Le Merrer, Didier Neveux

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

SONDe, Self-Organizing Replica Placement in Large-Scale Dynamic Systems

Vincent Gramoli and Anne-Marie Kermarrec and Erwan Le Merrer and Didier Neveux

N° 6052

Novembre 2006

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--9999--FR+ENG

SONDe, Self-Organizing Replica Placement in Large-Scale Dynamic Systems

Vincent Gramoli * and Anne-Marie Kermarrec † and Erwan Le Merrer ‡ and Didier Neveux §

Thème NUM — Systèmes numériques
Projet ASAP

Rapport de recherche n° 6052 — Novembre 2006 — 24 pages

Abstract: Initially introduced in the context of file sharing systems, the peer to peer communication paradigm goes far beyond and may be applied to a wide spectrum of distributed applications. The scalability of peer to peer applications relies both on an even distribution of the load between peers and the ability to react to system dynamics.

In this paper, we present the design, analysis, and evaluation of SONDe, a simple fully decentralized replica placement algorithm. Given an *object* (service or data), SONDe provides a peer with a constant upper bound on the number of logical hops to access a replica holder (*provider*), thus making tunable and predictable the communication latency between a peer and any replica (if used with logical-physical layer mapping algorithms). In addition, SONDe is able to adapt the number of replicas dynamically to reflect load variations in localized portions of the system. Each peer decides individually whether it holds a replica, based on the observation of its local neighborhood. We show theoretically that SONDe converges and provides an independent-dominating set of providers. Finally simulation results, conducted over different network topologies, demonstrate the efficiency of the approach.

Key-words: Large-scale dynamic overlays, Replica placement, Availability, Search bounds, Independent-Dominating set, Self-Organization, load handling

* vgramoli@irisa.fr
† akermarr@irisa.fr
‡ erwan.lemerrer@orange-ftgroup.com
§ didier.neveux@orange-ftgroup.com

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

SONDe, Self-Organizing Replica Placement in Large-Scale Dynamic Systems

Résumé : Présenté à l'origine dans le contexte des applications de partage de fichiers, le paradigme de communication pair-à-pair a évolué et peut être aujourd'hui appliqué à un large spectre d'applications distribuées. Le passage à l'échelle des applications P2P repose à la fois sur une distribution juste de la charge des requêtes entre les pairs du réseau et sur la capacité du système à réagir à la dynamique.

Nous présentons dans ce papier le design, l'analyse et l'évaluation de SONDe, un algorithme simple et totalement décentralisé de placement de réplicas. Donné un *objet* (fonction ou donnée), SONDe fourni à chaque pair une borne constante sur le nombre maximal de sauts logiques à effectuer pour accéder au détenteur d'un réplica (*fournisseur*), rendant ainsi paramétrables et prévisibles les latences de communications (dans le cas d'une utilisation conjointe avec des algorithmes de convergence des couches physique/logique). SONDe est également capable d'adapter le nombre de réplicas dynamiquement pour refléter les variations de charge dans les zones concernées du réseau. Chaque pair décide individuellement s'il doit héberger un réplica, et ceci basé sur l'observation de son voisinage. Nous montrons de façon théorique que SONDe converge et fourni un ensemble Indépendant-Dominant de fournisseurs. Enfin, des résultats basés sur des simulations conduites sur différentes topologies réseau démontrent l'efficacité de l'approche.

Mots-clés : Réseaux logiques dynamiques large échelle, placement de réplicas, disponibilité, bornes de recherche, ensemble Indépedant-Dominant, auto-organisation, répartition de charge

SONDe, Self-Organizing Replica Placement in Large-Scale Dynamic Systems

December 6, 2006

1 Introduction

Motivations A peer to peer (P2P) overlay network organizes a large set of participants (*nodes*), in a logical network on top of a physical topology. The scalability of P2P networks relies on the fact that each node may act both as a client and a server. Therefore, the number of potential servers grows linearly with the size of the system. For a given application to provide a service, a subset of nodes may be elected to act as servers, to which nodes can forward requests. One of the major challenges in P2P systems is how to place objects of a given service at some provider making the object rapidly accessible. This problem is known as the replica placement problem. Previous work on replica placement relies on a static placement of replicas assuming that client access patterns can be known in advance [10, 14, 3, 19]. However, to face the high dynamism of peer to peer systems, dynamic and adaptive replica placement is required. Several issues shall be considered to provide an efficient replica placement algorithm.

First, finding and accessing a replica in a P2P system should limit communications costs. For the client sake, an efficient replica placement algorithm should leverage the physical and/or logical network proximity so that client nodes direct their request to close replicas therefore limiting both time and communication complexity. However, holding a replica, and therefore serving client requests is a consuming task both bandwidth and computing power-wise. Therefore, the goal of such an algorithm is to limit the number of nodes holding a replica and yet to make sure the required guarantees are met.

Second, another crucial aspect is related to the amount of information one can use to implement the algorithm. Gathering global information in a P2P system is simply unrealistic due to the large-scale dimension of such systems but also to their dynamics which would

*vgramoli@irisa.fr
†akermarr@irisa.fr
‡erwan.lemerrer@orange-ftgroup.com
§didier.neveux@orange-ftgroup.com

require frequent and massive updates. Instead, in a P2P system, each node is aware of a small subset of other nodes, usually known as its neighbors. A scalable solution relies on the exploitation of the local knowledge of the system.

Contributions In this paper, we propose a dynamic and fully decentralized replica placement protocol, called SONDe (*Self-Organizing Network Density*) ensuring the availability of a replica in a constant number of logical (overlay) hops from any network location, while limiting as much as possible the number of providers of such a replica. Not only SONDe is able to provide such guarantees but also may be tuned to handle load variations in localized parts of the network. In the remaining of this paper, we assume that every node is able to act as a server, therefore becoming a service provider (holding a replica) is achieved by simply switching on this functionality. How to actually retrieve the last version of a replica is out of the scope of this paper.

SONDe provides guarantees in terms of number of logical hops and ensures in most configurations that each node will be able to access a replica, from any location of the network in a constant number of hops, at most h. SONDe is fully decentralized and highly scalable as nodes implement the replica placement algorithm based on the observation of their direct neighborhood in the overlay network. The intuition is as follows: each node observes its neighborhood and checks whether a replica can be found at most h hops away. If this is not the case, then it switches on its own functionality and becomes a provider. In order to control and bound the number of replicas, a provider able to reach another replica in h hops which is "older" than itself, takes the decision to switch its own functionality off. Theoretical results and simulations show that the distance (number of hops) required to access a replica is bounded by h and that the algorithm converges.

Even though the density of replicas maps the requirements, it might happen that some providers are overloaded. The extreme case to illustrate this situation is a star like topology where the central node would ensure that the guarantees are met but yet is too overloaded to actually serve the clients. To overcome this issue, each provider node in SONDe may take actions to adapt to such situations and decrease the parameter h locally. Therefore, SONDe, in a network-independent fashion, balances the load automatically between the system nodes. If a burst of load occurs in a localized area of the overlay, the number of providers is automatically increased in this specific area. Conversely, when the load decreases the number of providers is reduced accordingly in order to limit the unnecessary resource consumption.

Finally, the paper shows theoretically that SONDe provides an h-independent-dominating sets of providers. An h-independent-dominating set of providers is a set of providers such that each node is either provider or at most at distance h from a provider, but not both. Independence limits the number of providers while dominance allows constant access time from any system node.

A main guideline for SONDe is to provide a simple algorithm, that could be implemented easily on any type of overlay, regardless of the underlying overlay structure.

Roadmap The rest of this paper is organized as follows. Section 2 provides some background on the replica placement problem. Section 3 introduces the system model and states the problem we address in this paper. Section 4 presents the SONDe algorithm. Section 5 provides analysis that shows that SONDe solves the h-independent-dominating set problem. We simulate results of SONDe in a large scale dynamic context and analyzes the results. Finally, we discuss some open issues in Section 7 before concluding in Section 8.

2 Background on distributed replica placement

Many works in the peer to peer, parallel distributed systems and graph theory address to some extent the replica placement issue.

Distributed Hash Tables (DHT) [15, 17, 16, 4] is a well-known solution to find an object (data or service) in a P2P system. In DHT-based solutions, nodes share a global identifier space and the same objects are isolated on providers of the same identifier subspace. That is, any node can convey request message to a node closer to the provider. Isolating objects presents, however, two limitations. First, when many requests to the same object are simultaneously received, congestion might occur in the targeted subspace. Second, a node might be located too far from the targeted object, in terms of hops for some stringent application requirements, and thus accessing the object might be costly. DHT-based replication have proven efficient in some contexts but may suffer from a limited flexibility.

On the theoretical side, obtaining small dominating set in a graph remains an important issue. The problem of finding the minimal dominating set has been shown to be NP-hard [5] and an easier problem is to find a small (but not minimal) dominating set. In [9], the authors propose an algorithm that constructs small dominating set in a logarithmic number of rounds, while in [11], the authors construct a small dominating set in constant time. Importance of dominating set for P2P systems have been outlined by some specific applications [20]. These authors aim at maximizing the number of responses to an object search in a P2P system. This approach does not focus on providing the object at a constant number of hops. Finally, in [2] the authors focus on both independent and dominating sets, and propose a centralized solution to this problem.

Replica placement has been widely studied in the past years. Various works proposing Content Distribution Networks use tree-structured topologies like in [10]. The HotSpot [14] and HotZone [19] propose a solution to place replicas of any object at the nodes generating the highest traffic among all. Sortie [8] is providing a self organizing replication mechanism based on the demand for a service; the replicas are also placed based on the requests. Sortie does not take into account max hop bounds or latency limits between requesters and providers. Other papers [7, 3] addresses other but related issues: the number of centers and the centers placement. The former one consists of determining the number K of replicas needed while the latter consists of determining the ideal locations of those K replicas.

3 Design Rationale

3.1 System Model

The system consists of a connected set of n nodes. [1] Each node is uniquely identified and does not maintain any global information about the system. Nodes can join and leave the system at any time. The P2P overlay network is represented as a communication graph, denoted $\mathcal{G}$, in which a node i can communicate directly with a subset of nodes called its direct neighbors. More generally, we refer to the *neighbors* of i, as the set of nodes, $\mathcal{N}_i^h$, located at distance h from i, where $h \in \mathbb{N}^+$ is a constant. In other words, the minimal distance between i and any $j \in \mathcal{N}_i^h$ is lower than or equal to h. $\mathcal{G}$ is an undirected graph such that communication links are symmetric, formally if $i \in \mathcal{N}_j^h$ then $j \in \mathcal{N}_i^h$.

We call a peer not holding a replica, a *client* and a peer, holding a replica, a *provider*. A service or a data used in the system is equally referred to as an *object*. Objects are replicated over all system nodes and a copy of this object is called a *replica*. In the remaining, we only focus on a single replicated object. We refer to *providers* as the nodes providing this object.

Each peer hosting a replica has finite computing resources; too many simultaneous access queries may thus saturate it. A saturated replica will trigger a mechanism to distribute load by creating other replicas.

The communication delay between two neighbors is bounded by δ.

3.2 Problem Statement

The algorithm presented in this paper addresses the issue of limiting the number of providers of a single object x while providing any node with a object replica located nearby (in the h-vicinity). Providing the object consumes computation resources, limiting the replication of an object over the network saves resources for other objects. Second, providing any node with a replica at a small distance helps finding it and accessing it quickly. To formalize those two challenges we borrow two well-known definitions from graph theory: *independent* and *dominating* sets.

The *independence* notion expresses the idea of restricting the number of providers. Consider an undirected connected graph $G = \langle V, E \rangle$. A subset of vertices $\mathcal{I} \subseteq V$ of the graph G is called an *independent set* if and only if there is no edge in E between two vertices of $\mathcal{I}$. Observe that a simple solution to this problem is $\mathcal{I} = \emptyset$. Consider the communication graph $\mathcal{G}$. If the set of providers forms an independent set of $\mathcal{G}$, then the object does not consume computation resources of all nodes. Next, we recall the formal definition.

Definition 3.1 (Independent Set)**.** *Let $\mathcal{G} = \langle V, E \rangle$ be an undirected connected graph. A subset of vertices $\mathcal{I} \subset V$ is an* independent set *iff*

$$\forall (u, v) \in E \Rightarrow u \notin \mathcal{I} \vee v \notin \mathcal{I}.$$

[1] The value n corresponds to an instantaneous observation and can vary over time.

The *dominance* notion expresses the notion of availability. Consider an undirected connected graph $G = \langle V, E \rangle$. A subset of vertices $\mathcal{D} \subseteq V$ of the graph G is called a *dominating set* if and only if it exists an edge between any vertex of $V \setminus \mathcal{D}$ and a vertex of $\mathcal{D}$. Observe that a solution to this problem is $\mathcal{D} = V$. Like above, consider the communication graph $\mathcal{G}$. If the set of providers of x is dominant in $\mathcal{G}$, then x is easily available from any location. The formal definition follows.

Definition 3.2 (Dominating Set)**.** *Let $G = \langle V, E \rangle$ be an undirected connected graph. A subset of vertices $\mathcal{D} \subseteq V$ is a* dominating set *iff*

$$\forall u \in V \setminus \mathcal{D}, \exists v \in \mathcal{D} \text{ s.t. } (u, v) \in E.$$

Traditionally, those definitions apply to neighbors that are located at a single hop from the source. Depending on the application requirements, this might be too restrictive. Here, we relax such a constraint by letting the programmer define an edge between two nodes if their distance is smaller than a constant number h of hops. Consequently, the parameter h makes the replication degree and the availability degree of object x tunable.

Let an *independent dominating set* be a set $\mathcal{S}$ satisfying both Definitions 3.2 and 3.1. As said before, we extend the aforementioned definition to the h-neighborhood. Let E^h as the set of path composed of at most h consecutive edges of E. We do not mention definitions of *h-independent set* and *h-dominating set* since they rely simply on Definition 3.1 and Definition 3.2, respectively, with the augmented set of edges E^h. The formal definition is the following.

Definition 3.3 (h-Independent-Dominating Set)**.** *Let $G = \langle V, E \rangle$ be an undirected connected graph. A subset of vertices $\mathcal{S} \subset V$ is an* h-independent-dominating set *iff*

$$\begin{cases} \forall (u, v) \in E^h \Rightarrow u \notin \mathcal{S} \vee v \notin \mathcal{S}, \\ \forall u \in V \setminus \mathcal{S}, \exists v \in \mathcal{S} \text{ s.t. } (u, v) \in E^h. \end{cases}$$

Another interesting challenge would be to solve the Minimal Dominating Set (MDS) problem. Nevertheless, this well-known problem is NP-hard [5]. For the sake of solution efficiency, we rather focus on the independent dominating set problem. When a single node is added to a graph with an independent-dominating set, it is likely that no global update is necessary to retrieve an independent dominating set. Consequently, even though we do not ensure maximal independent set or minimal dominating set, our solution guarantees availability with a relatively low complexity. The goal of the SONDe algorithm is to provide at least one replica in the neighborhood of any node. To conclude, SONDe is a simple self-organizing replica placement algorithm making the set of providers to converge toward an h-dominating-independent property.

4 SONDe Algorithm

In this section, we first present the initial version of the SONDe algorithm, which implements a dominating independent set of providers in a fully decentralized way so that each node is

guaranteed to access a replica in at most h hops. Then we propose an extended version of SONDe, able to automatically adapt the replication degree to the load variations in localized parts of the network.

Each node decides locally whether to become a provider or not. For this purpose, every node i communicates periodically (with period $\rho \leq \delta$ or frequency $\tau = \frac{1}{\rho}$) with all nodes of its neighborhood $\mathcal{N}_i^h$, starting at its arrival on the overlay.

4.1 SONDe: *h-hops max access* replication algorithm

SONDe goal is to ensure that a node, at any time, can access a replica of a given service in its neighborhood $\mathcal{N}_i^h$ without global knowledge.

To converge towards this global property, each client in the system checks (called *function_check*), after ρ has elapsed, whether a replica can be reached in, at most, h hops.

In order to limit the associated overhead, only the replicas providers send back a `hello` message to a requesting node.

function_check leads to a state change of the checking node in the cases summarized on Figure 1.

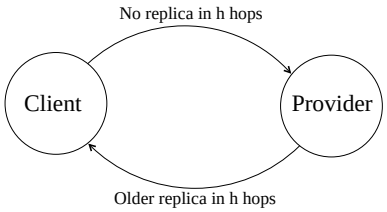

Figure 1: Basic SONDe state changes

- If the checking node is a client:
  - the node becomes a provider if no replica is found in $\mathcal{N}_i^h$
  - otherwise no action is executed.
- If the checking node is a provider:
  - nothing happens if no older replica is found in $\mathcal{N}_i^h$
  - otherwise, the current node switches off its replica functionality and becomes a client.

The age refers to the date at which the replica functionality was set. This info is piggybacked in the `hello` message sent to the requesting node.

The particular case where two peers in the same h-hops neighborhood have the same exact arrival date is provoking conflictual *function_checks* that are handled in a simple way

as follows: two regular peers will become provider after a first check, if no provider is already in their $\mathcal{N}_i^h$; during the second one, as the two providers have the same ρ, the discriminatory factor is the unique ID (for example the replica with the greater ID is switched off). After this last step, the fact that *function_checks* are done at the same time is no longer an issue.

This simple organisation leads each peer to have at least one replica in its neighborhood $\mathcal{N}_i^h$, thus reaching the dominating set. The independent set is observed when there is no more replica deletion, as no provider sees another one in its neighborhood. In the case where a peer has more than one provider in its neighborhood, the selection may rely on various parameters such as providers occupation state or minimum communication latency for example.

So far, we have made the assumption that the providers could, at any time, provide the service to any peer in their neighborhood. In practice, as peers have finite computing resources, some providers may be overloaded if too many peers, connected to the same provider, use the service simultaneously. This situation has a higher probability of occurring if h is high, as the provider has to handle more peers than at a lower level. To prevent service unavailability, we now introduce an extension of the base algorithm taking into account load variations.

4.2 *Load reactive* replication algorithm

The basic idea is that although the h-Independant-Dominating set property is ensured, some nodes might not be able to cope with request load. The pseudocode of the load reactive algorithm is provided in Algorithm 1. A provider may be overloaded if its connected clients are using too heavily its ressources. Such load pattern may affect the application as the provider may not be able to provide additional resources to its current clients or resources to new requesting clients. To avoid this situation, in addition to the h-hops parameter of the previous algorithm (global system max bound), we associate a second parameter $nbHops$ with each peer. $nbHops$ represents the peer current search bound ($nbHops \leq h$), and varies facing load variations as described bellow.

We assume that the occupation rate of the computing resources of a provider is represented as a percentage, returned by the peer operating system. In addition, two parameters are added to the peer hosting a replica: *underloadedThreshold* and *overloadedThreshold* ($0 < underloadedThreshold < overloadedThreshold < 100$). They indicate respectively the load threshold over which a replication mechanism should be triggered and under which replicas could be suppressed to free wasted resources:

- If the occupation rate of a provider remains over *overloadedThreshold* during a predefined period of time (*maxOverloadedRounds*), the provider launches the following process, leading to other replica creations: it notifies a fixed constant number of direct neighbors so that each neighbor changes its *nbHops* with the provider current one decremented by one; so does the provider itself. Each reached peer forwards the order to that constant number of direct neighbors, with a depth equal to *nbHops* from the originating overloaded peer. The next check, some of the nodes which *nbHops* has

been modified are likely to create replicas as they are not able to find other replicas at this reduced level. This action has a global consequence on the increase of the number of replicas in an overloaded area.

- On the contrary, a provider that remains under a fixed threshold (*underloadedThreshold*), for a given period (*maxUnderloadedRounds*), notifies its neihborhood at *nbHops* (as in the overloaded case) to set their *nbHops* with its own, incremented by one, and does the same. This process naturally decreases the number of providers in the former provider neighborhood.

When a response to a load event should occur, the heuristic does not simply tells $\mathcal{N}_i^{nbHops}$ to change their *nbHops* (but instead each peer forwards this order to a constant number of direct neighbors). In the case of a highly connected overlay, many peers received a change order, increasing convergence time. That constant can be set for example to approximatively $log(N)$. The aim of *maxUnderloadedRounds* and *maxOverloadedRounds* values is to create a *hysteresis* phenomena and stabilize the replica distribution by avoiding immediate duplication or deletion based on temporary non representative load events.

With the load handling heuristic, and during the *function_check*, a peer switches off it replica capability only if it finds in $\mathcal{N}_i^{nbHops}$, another replica with the same *nbHops* parameter, in order to preserve the previous *nbHops* changes; contrarywise, a peer does not create a replica even if another one is found with a different *nbHops* parameter. This keeps the locality of the load variation events, and thus does not disturb the balance of the replica distribution in the neighborhood. These phenomenons are detailed in Section 6.

Note that when *nbHops* changes, the new value given by the replica to the nodes reached is based on the one of the replica. The solution consisting in simply decreasing each node current value leads to unstable replica creation: in an overloaded region, where peers have more than one replica in their *nbHops* range, several change orders from the providers could really quickly set the *nbHops* value of the peers to 0, triggering too much replica creations (a peer at 0 obviously directly become a replica). Due to their number, those providers often reach an under load case by sharing local load among then, bringing on suppressions, and so on, increasing convergence time.

Back to a homogeneous distribution After SONDe has handled load variations in affected parts of the network, the replica distribution is no longer homogeneous. The orders of changing the value of *nbHops*, sent by the replicas to their neighbors, are non symmetrical in the sense that replicas with a high *nbHops* value obviously reach more peers than in the reverse case. This leads to the situation where when a network load occurs thus triggering replica creations, many peers get a lowered *nbHops*. When the load decreases, even if there are more replicas in the network due to the previous load handle, the sum of peers reached by the successive increases of *nbHops* is inferior to the total number of peers which *nbHops* has previously been modified. Therefore, some peers may stay at a *nbHops* value that is not representative of the network load anymore; that is why after a number of consecutive *nbHops* unchanged, the peer could simply query the current value of their neighbors and take

the average as new *nbHops*. This simple heuristic allows all the network peers to smoothly converge toward the max search bound h when there is no or a low load on the overlay.

5 Convergence to an h-Independent-Dominating set of Providers

In this section, we show that the SONDe algorithm converges.

Here are some preliminary notations for the proof. We refer to *c-providers* (standing for *confirmed providers*) as the providers that set their state to provider at least $\Delta > 2\delta$ time ago and that did not revert their state since then. Let $\mathcal{L}$ be the set of all legitimate configurations where the communication graph $\mathcal{G}$ admits a set $\mathcal{S}$ of c-providers such that $\mathcal{S}$ forms an h-independent-dominating set. Next, we define two specific subgraphs in which the h-independent-dominating property does not hold. Let a *provider-graph* (resp. *client-graph*) be a subgraph of $\mathcal{G}$ such that its nodes are a set of connected providers (resp. clients) of $\mathcal{G}$, its edges are the edges of $\mathcal{G}$ between those nodes, and it is maximal (i.e. no other provider-graph and client-graph, respectively, contains it). By abuse of notation, we say that a graph is h-independent-dominating if and only if the set of its providers represent an h-independent-dominating set.

Assume that the system stabilizes such that dynamics stop and providers are neither overloaded nor underloaded. Consequently, for any node, *nbHops* does not change over time and remains equal to h. We show that starting from any possible configuration, the algorithm converges to a configuration of $\mathcal{L}$. The following Lemma shows that a non-independent subgraph converges to an h-independent-dominating set.

Lemma 5.1. *A subgraph $\mathcal{P}$ of l nodes that is provider-subgraph becomes h-independent-dominating at the latest at $2\delta l$ time after stabilization.*

Proof. The longest convergence time is reached when all nodes react simultaneously: they all send messages and receive them at the same time, they all discover providers in their neighborhood... Indeed, a provider-subgraph $\mathcal{P}$ contains smaller provider-subgraphs when larger than two nodes. Consequently, included subgraphs can be solved in parallel which might speed up the convergence to the h-independent-dominating set (and at least not slow it down). We focus on the worst case that is when all nodes react simultaneously. By the tie-breaker node ID, all nodes of $\mathcal{P}$ become clients except the node i that owns the largest ID. Later on, the neighbors of i remain client while other nodes switch back simultaneously to the provider state, and so on. Clearly, the worst case scenario is when $\mathcal{P}$ is a line of size l where nodes are ordered with increasing IDs. Since the delay to send a message is upper bounded by δ so as its transmission delay, this leads to an h-independent-dominating set at the latest at time $2\delta l$ after stabilization time. □

Next Lemma says that an initially non-independent subgraph that becomes h-dominating-independent remains h-dominating-independent.

Lemma 5.2. *If a provider-subgraph $\mathcal{P}$ becomes h-independent-dominating, its c-providers form an h-independent-dominating after a finite amount of time.*

Proof. On the one hand, when a provider becomes client, one of its neighbor j outside of $\mathcal{P}$ (and originally a client) may switch to the provider state. However, this potential change does not affect the rest of the graph $\mathcal{P}$ and the conflicting region induced by $\mathcal{P}$ is still solved. On the other hand, by definition of the provider-subgraph, any neighbor k (outside of $\mathcal{P}$) of a remaining provider of $\mathcal{P}$ can only be a client. Consequently, it will never switch its state to provider since it already knows a provider in its neighborhood. Finally, after an additional delay of Δ all providers become c-providers. □

The following Lemma states that a non-dominating subgraph converges to a stable h-independent-dominating set.

Lemma 5.3. *If a subgraph $\mathcal{C}$ is a client-subgraph at time $t = 2\delta n$ after stabilization, then its c-providers form an h-independent-dominating set at the latest at time $t + \Delta$.*

Proof. By definition, the client-subgraph is independent at time t. Moreover, Lemma 5.1 and Lemma 5.2 show that there is no non-independent subgraph in $\mathcal{G}$ at time t. There are two cases: either $\mathcal{C}$ is already a dominating set because all of its nodes are neighbors of some providers in $\mathcal{G}$ (but outside graph $\mathcal{C}$), or $\mathcal{C}$ is non-dominating. In the former case, the proof is straightforward, while in the latter case at least one node switches its state to become a provider. More precisely, after a timeout, some nodes become providers, say at time t. At time $t + \delta$ all have sent a message and by time $t + 2\delta$ they all have successfully received all messages sent by their neighbors. Consequently, at time $t + \Delta$, at most one provider (the one with the lowest ID) among all its neighbors becomes a c-provider and $\mathcal{C}$ becomes h-independent-dominating. □

Finally, the following Theorem concludes the proof that SONDe converges to an h-independent-dominating set after system stabilization.

Theorem 5.4. *SONDe converges.*

Proof. By Lemma 5.1 and Invariant 5.2, we know that an initially non-independent subgraph converges to a set of c-providers forming an h-independent-dominating set. Finally, Lemma 5.3 shows that a non-dominant subgraph converges also to a set of c-providers forming an h-independent-dominating set in a finite amount of time. To conclude this shows that the SONDe converges to a legitimate configuration. □

6 Evaluation

6.1 Experimental setup

To evaluate SONDe performance, we implemented SONDe using PeerSim [13], a discrete event/cycle based simulator, well-suited for large-scale networks. To assess the independence

of SONDe to the underlying structure, we created undirected unstructured peer to peer overlays using three different topologies. We do not model the queuing delays nor packet losses.

Topologies The Figures presented in the following of this paper are based on three types of well known topologies. Experiments were driven up to 1,000,000 nodes; for practical considerations the following results are based on 10,000 node networks. Scalability considerations are specifically addressed in Section 7.

The first topology we consider is a simple 2-dimensional overlay, where each peer is connected to 3 to 6 (average of 3.8) other peers that are "close" (in terms of the cartesian distance) from itself in the 2-D space. This topology is particularly useful to visually validate the homogeneous repartition of the replicas and to observe the impact of local load events on the rest of the overlay in terms of replication reaction. More generally, the mapping between the logical and physical topologies would minimize in practice the overall communication latencies in a real world implementation.

The second considered topology is a homogeneous random network, where each node has a constant number of neighbors, chosen uniformly at random. A node chooses another one as neighbor (selected uniformly at random) if the contacted node does not have already the maximum number of edges. Otherwise, the process keeps going until each node fills its view. We use a view of size 4.

The last topology is representative of the so-called *scale free* topologies and is based on the Barabási and Albert model [1]. Large scale networks with node degrees following a power law, as the Internet, are nowadays intensively used to validate practical approaches. Peers have between 2 and 230 neighbors, with an average of 3.99.

For the three considered topologies, each peer has $\log N$ neighbors on average, leading to an equivalent number of edges (around 20,000). This provides a fair comparison on the output results.

We call a *round* the time needed by all peers of the overlay to accomplish their *function_check*.

We evaluate the algorithms along three metrics: *(i)* the number of rounds required to converge to a stabilized configuration; *(ii)* the reaction to load variations in term of number of created replicas; *(iii)* the percentage of peers able to access a non overloaded replica under their current *nbHops* (satisfied peers), despite load variations, or constant arrival/departure of overlay nodes (*churn*).

6.2 *h-hops max access* replication algorithm

h (the predefined max hop bound) is set to 4 in the following simulations.

Figure 2 presents the convergence starting from a replica-free network to a stable number of replicas, under an extreme scenario where 20% of the overlay peers run their *function_check* simultaneously at each round (the rest of the peers act asynchronously). Despite such an extreme setting, only a few number of rounds are necessary to reach a stable state. Recall

that a stable state is reached when a h-independent-dominating set is achieved, thus any peer i has a replica in $\mathcal{N}_i^h$ and when no replica has another one in $\mathcal{N}_i^h$; As predicted by the proof of Section 5, the experimental analysis shows that SONDe converges. The cumulative distribution function presented on Figure 3 confirms this; a distance of 0 hop, simply means that the current peer is itself a replica and therefore uses its own replica when needed.

The left part of Figure 4 visually depicts the homogeneity of the replica distribution (bigger dots), on the 2-dimensional generated overlay, where peers are homogeneously distributed.

The second point highlighted in Figure 2 is that the number of created replicas, which satisfies the h-independent-dominating set, varies according to the underlying topology, even with a similar average of neighbors (and same number of edges). This is directly related to the diameter of the overlays: small diameter overlays trigger less replica creation than larger diameter overlays, because a *function_check* in $\mathcal{N}_i^h$ reaches more peers. Indeed, it is verified experimentally, starting from the scale free topology (lowest diameter overlay of our experiments) to the largest one (2-D): lowest diameter overlays, for an equivalent number of peers and connectivity, have higher degree nodes. These nodes act as *gateways*, as they provide their important set of neighbors and give access to more peers within the same hop-radius.

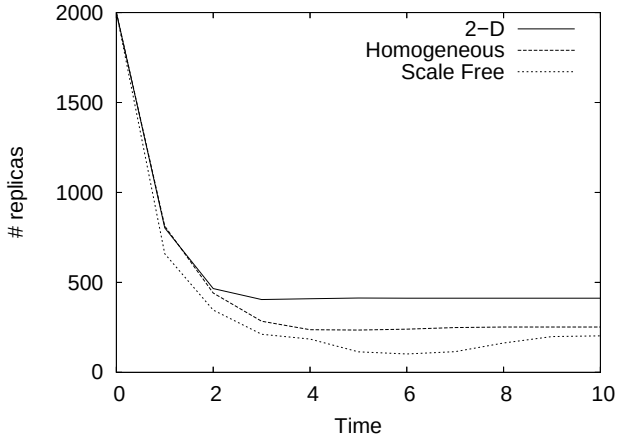

Figure 2: Convergence toward a stable state, 20% of peers simultaneously checking $\mathcal{N}_i^h$ to create/remove a replica, 10,000 2-D peer unstructured overlay

6.3 *Load reactive* replication algorithm

In order to simulate the load on each replica, we associate each replica with the number of maximal simultaneous access it can handle. We also apply a load pattern to the overlay,

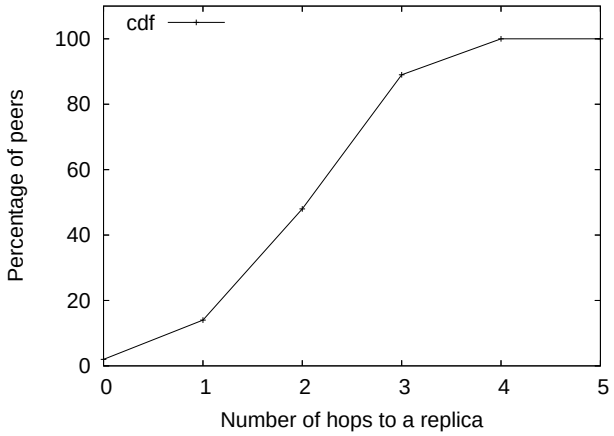

Figure 3: CDF of the distance between the peers and their replica, h=4, 10,000 2-D peer unstructured overlay

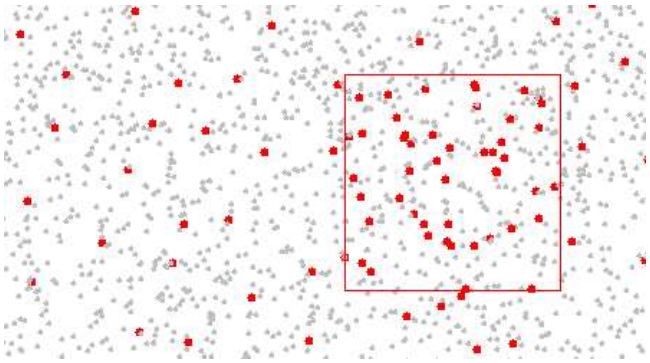

Figure 4: Distribution of the replicas on a 2-D overlay; Left: no load applied, Right overloaded region in the square overlay, h=4, 10,000 peer unstructured overlay

by increasing and decreasing the number of peers using the provided service. Figure 5 depicts the number of replicas created in response to the load variation (gradual increase from round 1000 to round 5500 where all peers are simultaneously querying a replica). We considered three replica capacities: 5, 10 and 25 active clients per replica. The number of clients that are likely to use a replica (client attached to this replica) is not upper bounded, as messages that simply check the presence or the availability of a replica are not resource consuming. *underloadedThreshold* and *overloadedThreshold* are set to 30% and 70% of the previous value, that is 3 and 7 peers simultaneously use the service as the maximum replica capacity is set to 10. As expected, when a replica is able to serve many peers, it takes more time to saturate it as illustrated by the evolution of the replication rate. When the load decreases, the reactive algorithm suppresses underloaded replicas for the sake of resources saving. Once load picks have disappeared, the system reaches a stabilized configuration quickly.

Reactivity highly depends on the parameters *overloadThreshold*. The lower its value, the more reactive the system is to load increase, preventing earlier replicas to be fully saturated. Obviously, this comes at the price of potentially more resource wastes, due to the creation of not fully used replicas.

In the rest of the experiments, we keep 10 active clients per replica. Figure 6 shows the increase of the number of replicas under the same load pattern, for the three topologies introduced earlier. *underloadedThreshold* is set to 10%. This value allows a quicker convergence towards a stable state (around t=6500) for the 2-D compared to Figure 5. This leaves more replicas in the system when load decreases as the load has to be very low to trigger replica deletion for resources saving.

We observe that the curves based on 2-D and homogeneous topologies evolve nearly similarly. The scale free curve is a bit noisier: the stabilization time is increased. This is due to the fact that the configuration of replicas placement is harder to reach on this topology in presence of an important number of replicas, as the neighborhood of each peer is wider.

We observe on Figure 7, a similar evolution of the load for the three topologies. We focus on the percentage of peers which are able to find a non overloaded replica in $\mathcal{N}_i^{nbHops}$. Note that this is a constraint, as the percentage is theoretically higher in $\mathcal{N}_i^h$ (recall that $nbHops \leq h$). We observe that the percentage of satisfied peers is close to 100% at any time, despite sudden increases. The unavailability peaks are noticeable in the scale free case, corresponding in large decreases of the number of replicas that can be observed on Figure 6.

This overall good behavior comes from the fact that the peers generally have several replicas in $\mathcal{N}_i^{nbHops}$ (thus have a greater chance to find a non saturated provider), coupled with the high reactivity of the replication algorithm.

An important parameter as far as load increase is concerned is to what extent the replication degree remains localized to the overloaded part of the network. We observed on the right hand side of Figure 4 that the increase of replica density is circumvented to the affected area only. Replicas are naturally created in the loaded zone, while no impact on the rest of

the overlay is observed. More specifically, a load event in a part of the overlay is absorbed by its direct neighborhood and thus do not trigger, as a side-effect, a global reorganization of the overlay replicas over the system.

Churn To illustrate the behavior of the algorithm under churn, we impose a constant peer arrival and departure in the system. Stutzbach et al. [18] highlighted that session lengths in P2P system are consistent across systems such as Gnutella, Kad, BitTorrent: almost all nodes have left after one day. In order to apply a realistic churn in our cycle-based simulations, we take the session length for half of the peers; after 30 minutes, all the peers have left the overlay, constantly replaced by new ones. A cycle in our simulations correspond to 1 second, we have thus a churn of 0.055% the peers per round. When the number of neighbors of a peer falls under the predefined minimum value, the peer looks for another contact peer as new neighbor. Figure 8 shows the impact of such a configuration on the proportion of node having a replica in their h-vicinity in the three considered topologies. The curves present the percentage of peers that have found a replica within $\mathcal{N}_i^h$, starting from a network replica-free. We observed that only a few rounds are required in make the rate of unsatisfied peers to lie within 1%, regardless of the topologie. The churn is stopped at round 1000 and we observe that the system converges quickly to a stable configuration in which all peers are satisfied.

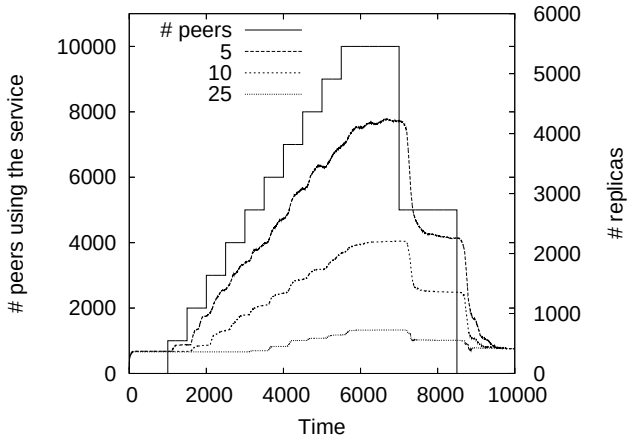

Figure 5: Replica creations in reaction to load varaiations, under different replica capacities, h=4, in a 10,000 node, 2-D unstructured overlay

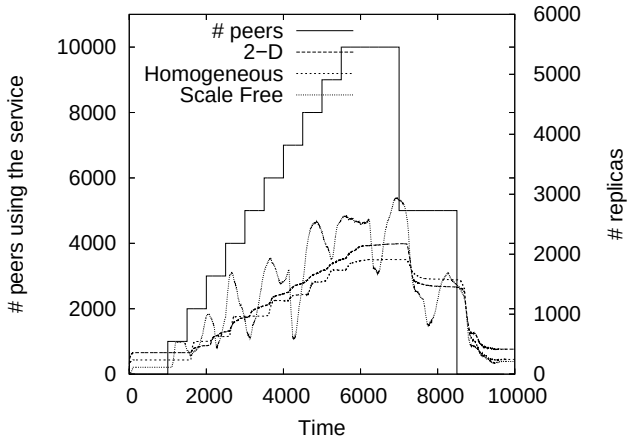

Figure 6: Evolution of the number of replicas in the overlay under replica usage variation, h=4 in a 10,000 node unstructured overlay

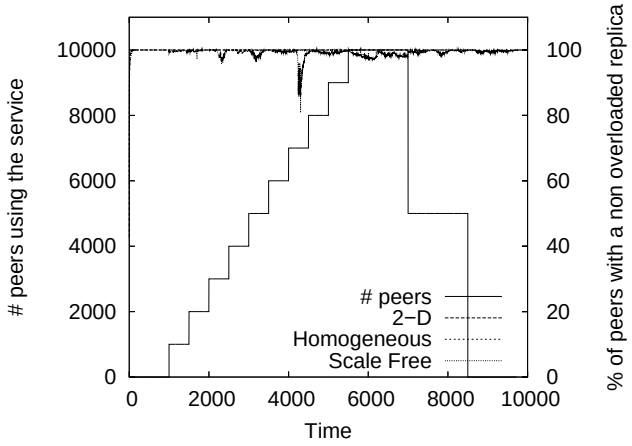

Figure 7: Percentage of satisfied peers (able to reach a replica within *nHops*) under usage variation, h=4 in a 10,000 node unstructured overlay

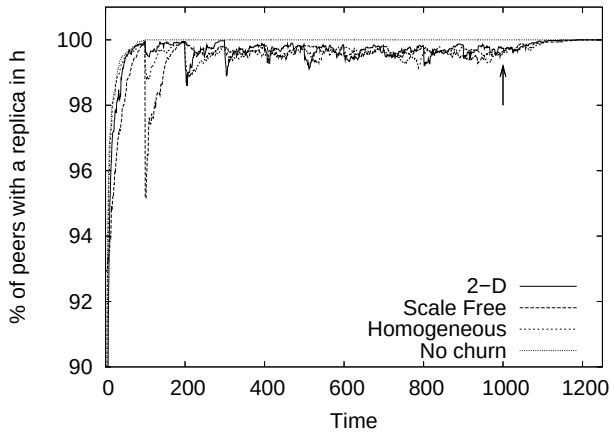

Figure 8: Percentage of satisfied peers (able to reach a replica within $\mathcal{N}_i^h$) under constant peer arrivals and departures (Churn stopped at t=1000)in a 10,000 node unstructured overlay

7 Discussion

SONDe has been designed with simplicity in mind and does not require any specific overlay structure. We presented the algorithm in the context of a single service duplication. We believe however that several replicas could be launched in parallel in order to provide the access to different services or data on the same overlay network. In this section, we discuss some of the open issues related to the deployment of SONDe.

Scalability Although we presented the results obtained in a 10,000 nodes in the paper, we scaled up the simulation to 100,000 and 1,000,000 peers in the 2-D topology. The number of created replicas on a non loaded network was respectively 2434 and 24363 (249 replicas on a 10,000 peers network for the plotted runs), and show a linear increase of the number of replicas in the size of the system.

Neighborhood exploration In this paper, we did not make any specific assumption on the exploring method used to search a replica in a peer neighborhood during a *function_check*. Several approaches could be considered differing by the traditional trade-off between latency and overhead. Basic search techniques such as flooding in unstructured peer to peer overlays, are expensive in terms of overhead [12], techniques like *Expanding Ring* [12] or *Random Breadth-First-Search* [6] with a fixed TTL could for example be used to significantly improve the overall performance. As an alternative, in order to reduce the number of messages, replicas could advertise their presence in $\mathcal{N}_i^h$, carrying a distance information increased at each hop, in order from their neighbors to evaluate if the $\mathcal{N}_i^{nbHops}$ condition is satisfied. Such

a reactive heuristic would avoid each standard peer to periodically perform a neighborhood check. Obviously the value of h impacts on the performance of the neighborhood exploration.

Adaptivity to system characteristics Our experiments showed that SONDe, regardless of the topology, provides a good resilience to churn. The plotted experiment (Figure 9) represents a worst case scenario, as it was driven on a network without peers accessing replicas (without load); the overlay thus contains a minimal replica number, compared to a loaded network where replicas are added to keep the accurate level of quality of service. The number of replicas created by the *Load reactive* replication algorithm thus increases the probability of finding a replica facing connectivity losses induced by churn. Part of the future work includes the study of the complexity of SONDe to target stringent 100% availability at any time.

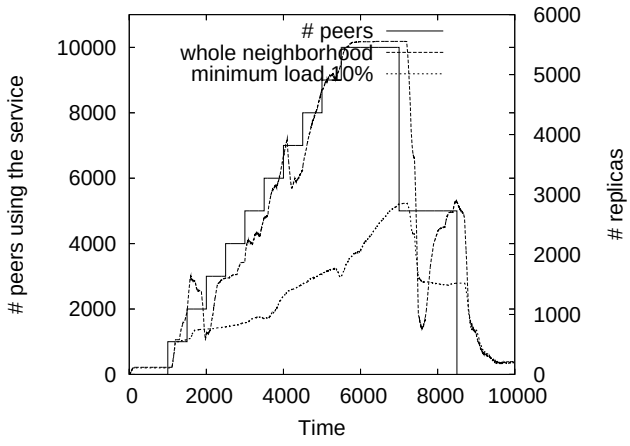

Figure 9: Evolution of the number of replicas in the overlay under service usage variation, on a 10,000 peer scale free overlay

In the context of low diameter overlays, represented in our experiments by the scale free graph, Figure 9 highlights two points:

- Without the bound on the number of direct neighbors to which an order of *nbHops*-change is forwarded (highest curve), the number of created replicas is more important. No mechanism for resource saving is triggered in the system, as the occupation rate of the replicas stays just above the minimal load threshold of 10%. The system thus reaches a stable state, but with a useless number of replicas, that justifies our heuristic.
- The second curve has to be compared to the scale free curve of Figure 6. The minimal load threshold is set to 10%, instead of 30%, thus triggering less replicas deletion.

This helps the system to react more smoothly to the load variations, and to decrease convergence time.

To conclude on parameter setting, the general idea is that the more we look for replication optimality, that is minimizing the number of replicas while accessing a sufficient level of quality of service, the higher the complexity, and the longer the convergence to reach such a state.

The fact that SONDe does not make any assumption on the underlying topology, because only reacting based on a neighborhood observation, it can thus be implemented on any type of overlay, and adapts gracefully to connectivity and load pattern changes. However, studying heuristics for specific network topologies could be an interesting future research direction.

8 Conclusion

In this paper, we have presented the design, analysis and evaluation of the replica placement algorithm SONDe for large-scale dynamic systems. SONDe provides a *h-hops max access* algorithm, intended to enable a peer to find a replica in a maximum of h hops away from itself. SONDe also implements a *load reactive* algorithm, intended to avoid the replica saturation under service load variations. SONDe ensures the automatic creation and deletion of replicas following the load variations in the network. The algorithms ensures that the impact remains limited to the network area affected by the load variation, while preserving the max search bound for any peer to a service provider. Simulations results demonstrate the ability of the algorithms to adapt to variations occurring in the peers environment. To conclude, SONDe is a simple and efficient system to distribute a set of highly accessed replicas, in a fully self-adaptive fashion regardless of the underlying network infrastructure.

References

[1] R. Albert. *Statistical mechanics of complex networks*. PhD thesis, University of Notre Dame, Notre Dame, Indiana 46556, 2001.

[2] Paola Alimonti and Tiziana Calamoneri. Improved approximations of independent dominating set in bounded degree graphs. In *Workshop on Graph-Theoretic Concepts in Computer Science*, pages 2–16, 1996.

[3] Yan Chen, Randy H. Katz, and John D. Kubiatowicz. Dynamic replica placement for scalable content delivery. In *Peer-to-Peer Systems: First International Workshop, (IPTPS'02)*, pages 306–318, Cambridge, MA, USA, March 2002.

[4] Peter Druschel and Antony Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 75, Washington, DC, USA, 2001. IEEE Computer Society.

[5] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[6] Fletcher G.H.L., Sheth H.A., and Borner K. Unstructured peer-to-peer networks: Topological properties and search performance, 2004.

[7] S. Jamin, Cheng Jin, Yixin Jin, D. Raz, Y. Shavitt, and Lixia Zhang. On the placement of internet instrumentation. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'00)*, volume 1, pages 295–304. IEEE, 2000.

[8] H. Jamjoom, S. Jamin, and K. Shin. Self-organizing network services, 1999.

[9] Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. An efficient distributed algorithm for constructing small dominating sets. *Distrib. Comput.*, 15(4):193–205, 2002.

[10] Xiaohua Jia, Deying Li, Xiaodong Hu, and DingZhu Du. Optimal placement of web proxies for replicated web servers in the internet. *The Computer Journal*, 44(5):329–339, 2001.

[11] Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. *Distrib. Comput.*, 17(4):303–310, 2005.

[12] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. 2002 ACM SIGMETRICS conference (2-page poster)*, 2002.

[13] http://peersim.sourceforge.net/.

[14] Lili Qiu, V.N. Padmanabhan, and G.M. Voelker. On the placement of web server replicas. In *20th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings (INFOCOM'01*, volume 3, pages 1587–1596. IEEE, 2001.

[15] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[16] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[17] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[18] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Internet Measurement Conference*, Rio de Janeiro, Brazil, October 2006. Proceedings of ACM SIGCOMM/USENIX.

[19] M. Szymaniak, G. Pierre, and M. van Steen. Latency-driven replica placement. In *Proceedings of the Symposium on Applications and the Internet*, pages 399–405, 2005.

[20] Chunlin Yang and Jie Wu. Dominating-set-based searching in peer-to-peer networks. In *Grid and Cooperative Computing*, pages 332–339, 2003.

Algorithm 1 SONDe algorithm at node i

Prerequisite Functions:

setNbHops$(k)_i$ notifies its neighbors at nbHops and sets
the communication range of node i, $nbHops$, to k.
overloaded$()_i$, underloaded$()_i$, and provider$()_i$ return true if
the node i is overloaded, underloaded, or its status is *provider*,
respectively, false otherwise.
findProvider$(nbHops)_i$ returns true if there is a node in
$\mathcal{N}_i^{nbHops}$, false otherwise.
findOlderProvider$(nbHops)_i$ returns true if there is an older node
in $\mathcal{N}_i^{nbHops}$, *false* otherwise.
becomeRegular$()_i$ sets the status of node i to a non-provider.
becomeProvider$()_i$ sets the status of node i to *provider*.
getNbrAvg$()_i$ returns the average of the $nbHops$ values of the peer's neighbors

Initial State:

$h, nbHops \leftarrow 4$
$overloadedRounds, underloadedRounds, unchangedRounds \leftarrow 0$
$overloadedThreshold \leftarrow 0.7$
$underloadedThreshold \leftarrow 0.3$
$maxOverloadedThreshold \leftarrow 1$
$maxUnderloadedThreshold \leftarrow 2$
$unchangedThreshold \leftarrow 5$

Algorithm:

```
if ¬replica() then
  if ¬findProvider(nbHops) then
    becomeProvider()
  else
    unchangedRounds ← unchangedRounds + 1
  if unchangedRounds > unchangedThreshold then
    nbHops ← getNbrAvg()
else
  if findOlderProvider(nbHops) then
    becomeRegular()
  else
    if overloaded() then
      overloadedRounds ← overloadedRounds + 1
      underloadedRounds ← 0
    else if underloaded() then
      underloadedRounds ← underloadedRounds + 1
      overloadedRounds ← 0
    else
      overloadedRounds ← 0 // normal state
      underloadedRounds ← 0
    if overloadedRounds > maxOverloadedRounds then
      setNbHops(nbHops − 1)
      overloadedRounds ← 0
    else if underloadedRounds > maxUnderloadedRounds then
      setNbHops(nbHops + 1)
      underloadedRounds ← 0
```

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399